



Einführung in NumPy: Das Fundament für Data Science

Warum wir es brauchen, wie es funktioniert und warum es so schnell ist.

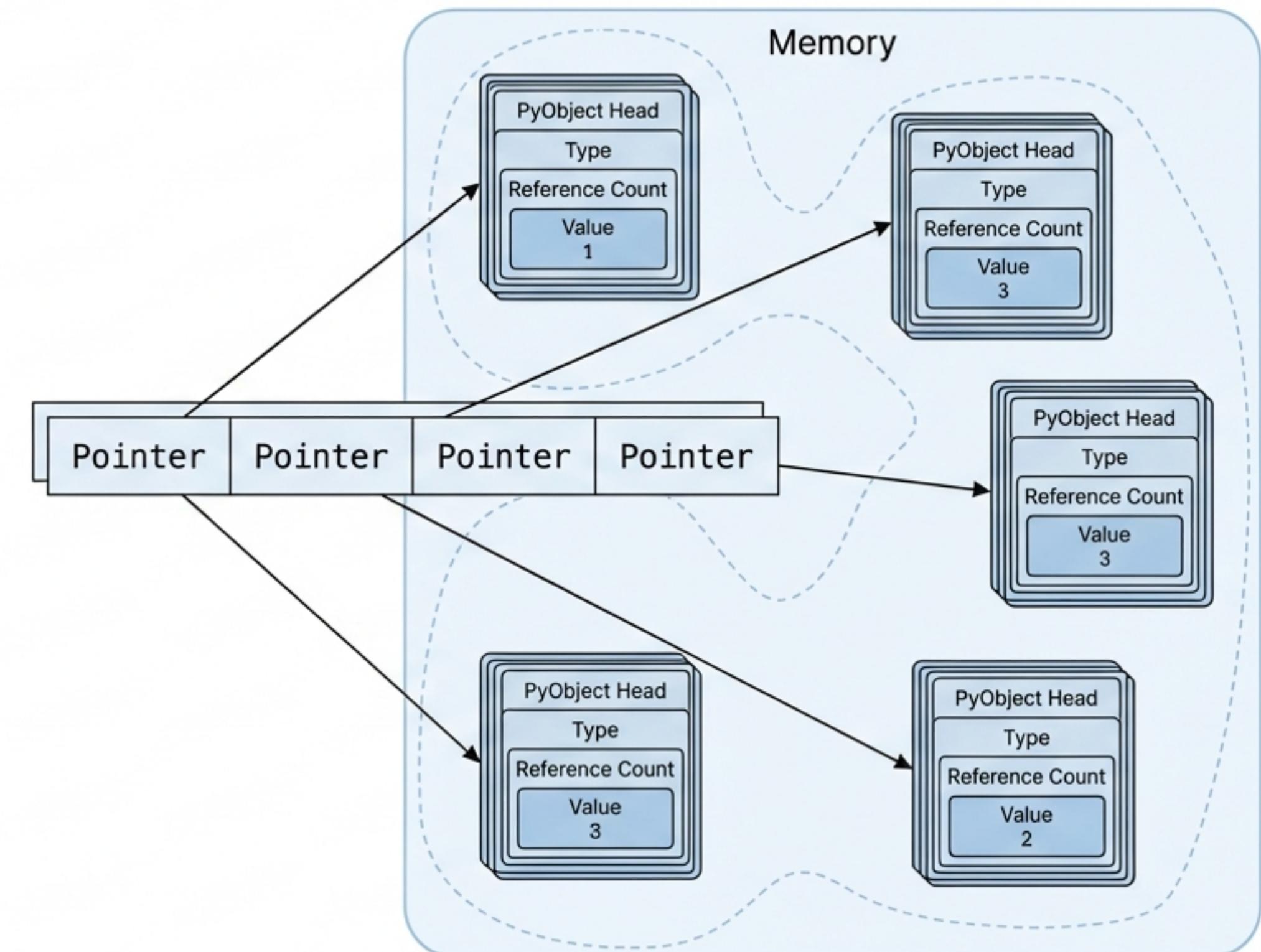
Basierend auf Inhalten aus 'Python Data Science Handbook' und 'Numerisches Python'.

Das Problem: Python-Listen sind flexibel, aber teuer

Python-Listen speichern Referenzen (Pointer) auf Objekte, nicht die Daten selbst.

Jedes Element ist ein vollwertiges Python-Objekt mit eigenem Overhead (PyObject (PyObject Head, Type, Reference Count)).

Resultat: Gestreute Daten im Speicher (Scattered Memory) führen zu ineffizientem Zugriff.

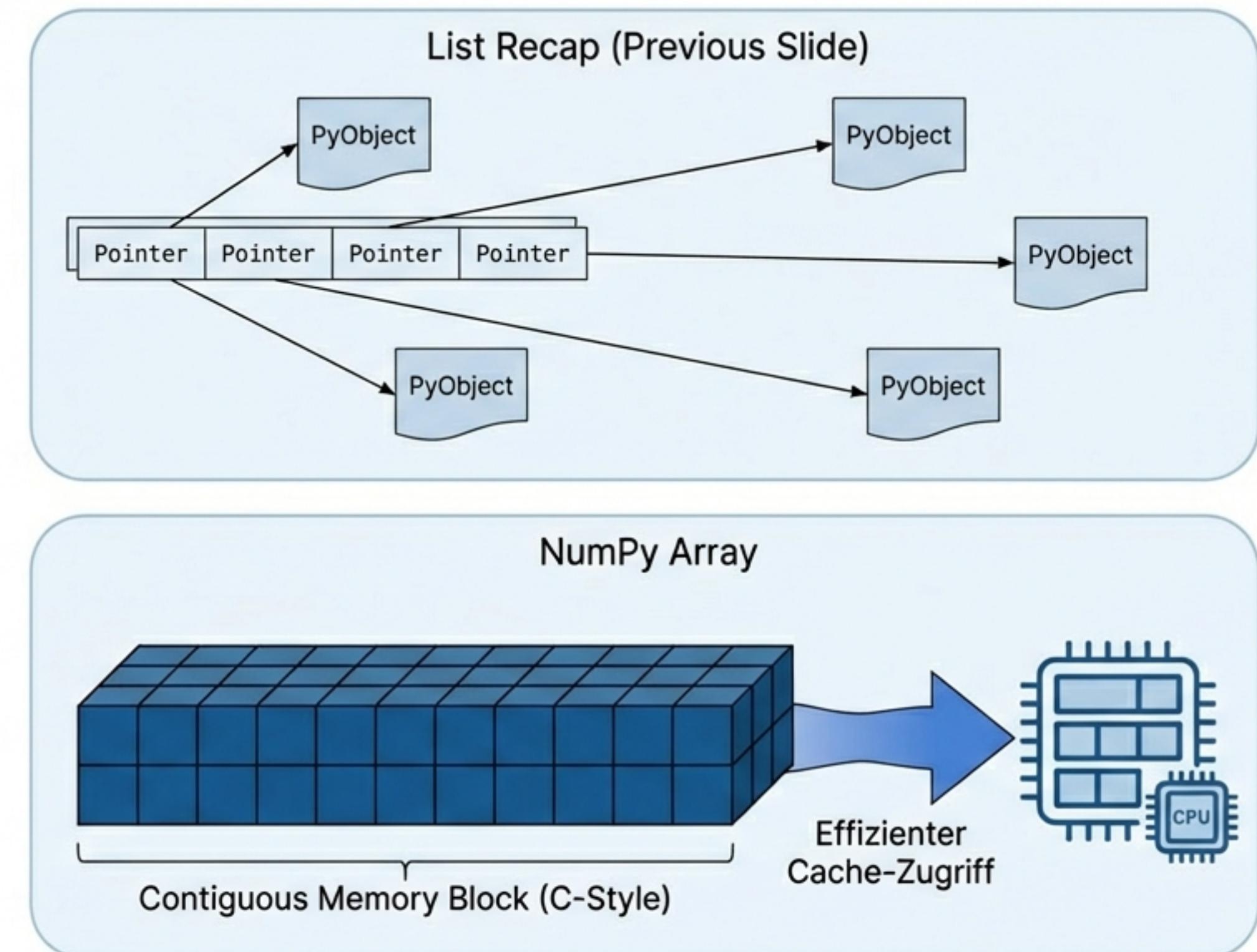


Die Lösung: Das NumPy ndarray

Homogene Daten: Im Gegensatz zu Listen müssen alle Elemente denselben Datentyp haben (z.B. nur 'int64').

Contiguous Memory: Daten liegen in einem einzigen, zusammenhängenden Speicherblock (wie in C oder Fortran).

Effizienz: Die CPU kann Datenblöcke vorhersagen und effizient in den Cache laden (Locality of Reference).



Warum ist NumPy schneller? Vektorisierung

SIMD (Single Instruction, Multiple Data)

Operationen werden auf den gesamten Array gleichzeitig angewendet, statt Element für Element. Keine Python-Schleifen: Die Iteration passiert in optimiertem C-Code, nicht im langsamen Python-Interpreter.

Python (Slow)



```
# Langsame Schleife
result = []
for x in liste:
    result.append(x * 2)
```

NumPy (Fast)



```
# Vektorisierte Operation
result = array * 2
```

Der Beweis: Performance-Benchmark

Berechnung von Kehrwerten ($1 / x$) für 1 Million Elemente.



"Mobiltelefone haben Gigaflops-Leistung, aber Python-Schleifen bremsen sie aus."

Erstellung von 1D-Arrays

Aus Liste

```
np.array([1, 2, 3])
```

1	2	3
---	---	---

Bereich (Integer)

```
np.arange(0, 10, 2)
```

0.0	0.25	0.5	0.75	1.0
-----	------	-----	------	-----

Bereich (Integer)

```
np.arange(0, 10, 2)
```

0	2	4	6	8
---	---	---	---	---

Hinweis: np.linspace beinhaltet das End-Element, np.arange nicht.

Initialisierung

```
np.zeros(5)
```

0.	0.	0.	0.	0.
----	----	----	----	----

Datentypen (dtype) – Die Basis der Geschwindigkeit

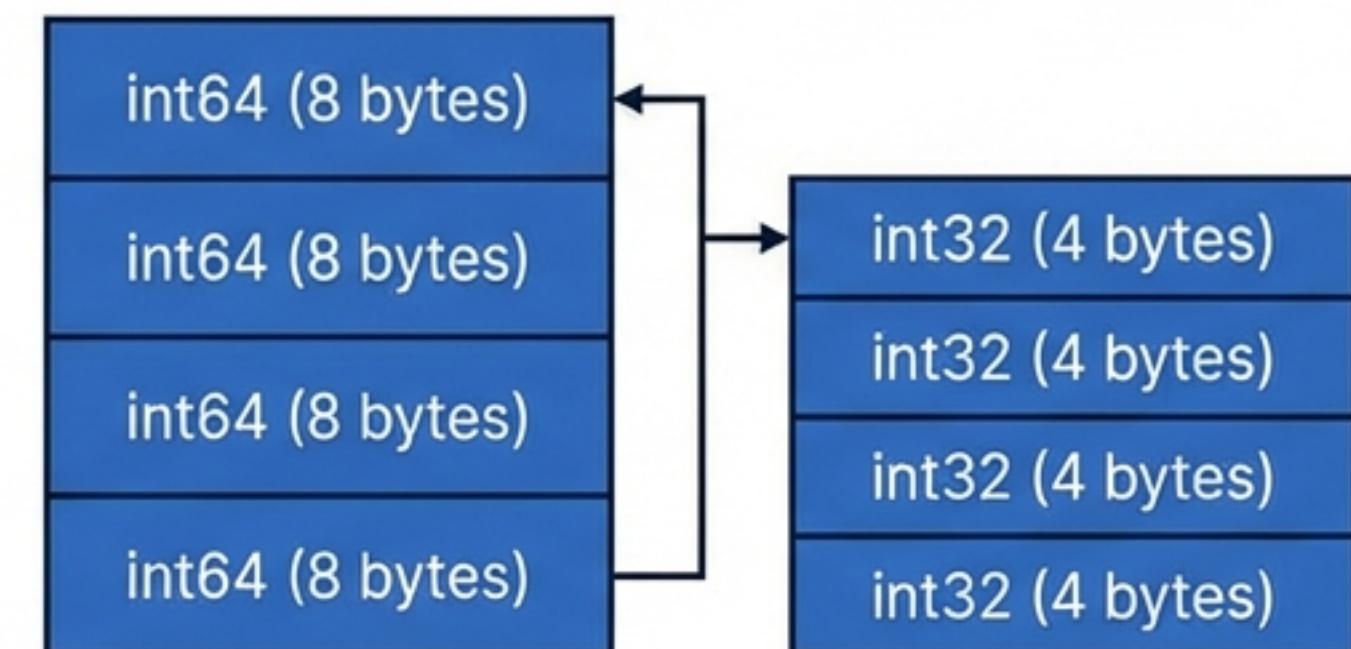
Da Speicher reserviert wird, muss der Datentyp fixiert sein. NumPy rät den Typen meist automatisch, kann aber explizit gesetzt werden.

int64 / int32: Ganze Zahlen (Standard)

float64: Fließkommazahlen

bool: Wahrheitswerte

```
x = np.array([1, 2, 3], dtype='float32')
print(x.dtype)
# Ausgabe: float32
```



Elementare Operationen (1D)

Arithmetische Operatoren wirken elementweise.

1. Scalar Addition:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|} \hline 5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 6 & 7 & 8 \\ \hline \end{array}$$

Broadcasting: 5 wird auf alle Elemente addiert

2. Element-wise Addition:

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 10 & 20 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 11 & 22 \\ \hline \end{array}$$

Addition an gleicher Position

3. Element-wise Multiplication:

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 4 & 5 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 8 & 15 \\ \hline \end{array}$$

Kein Matrixprodukt! Elementweise Multiplikation.

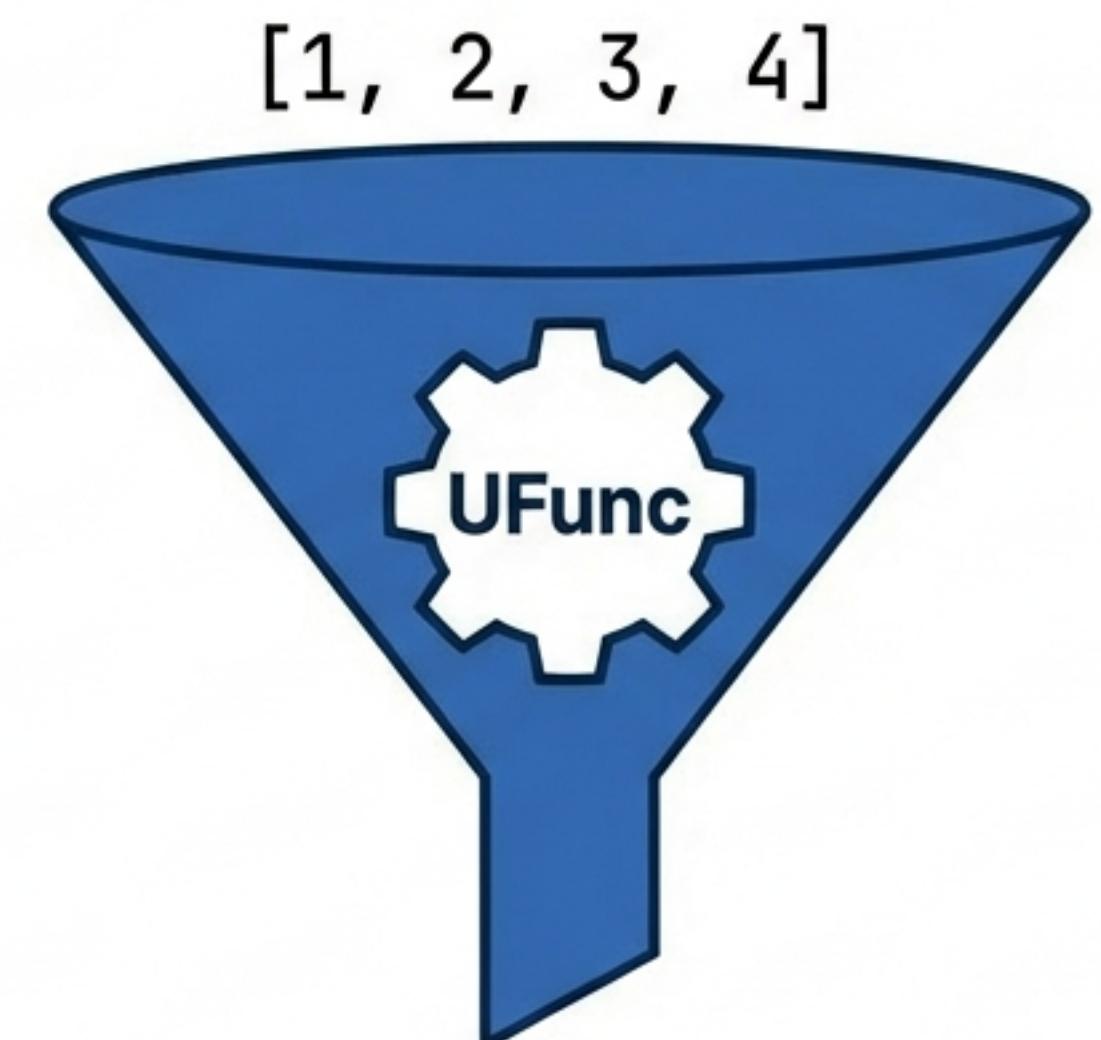
Universal Functions (UFuncs)

Schnelle, vektorisierte Wrapper für mathematische Routinen.

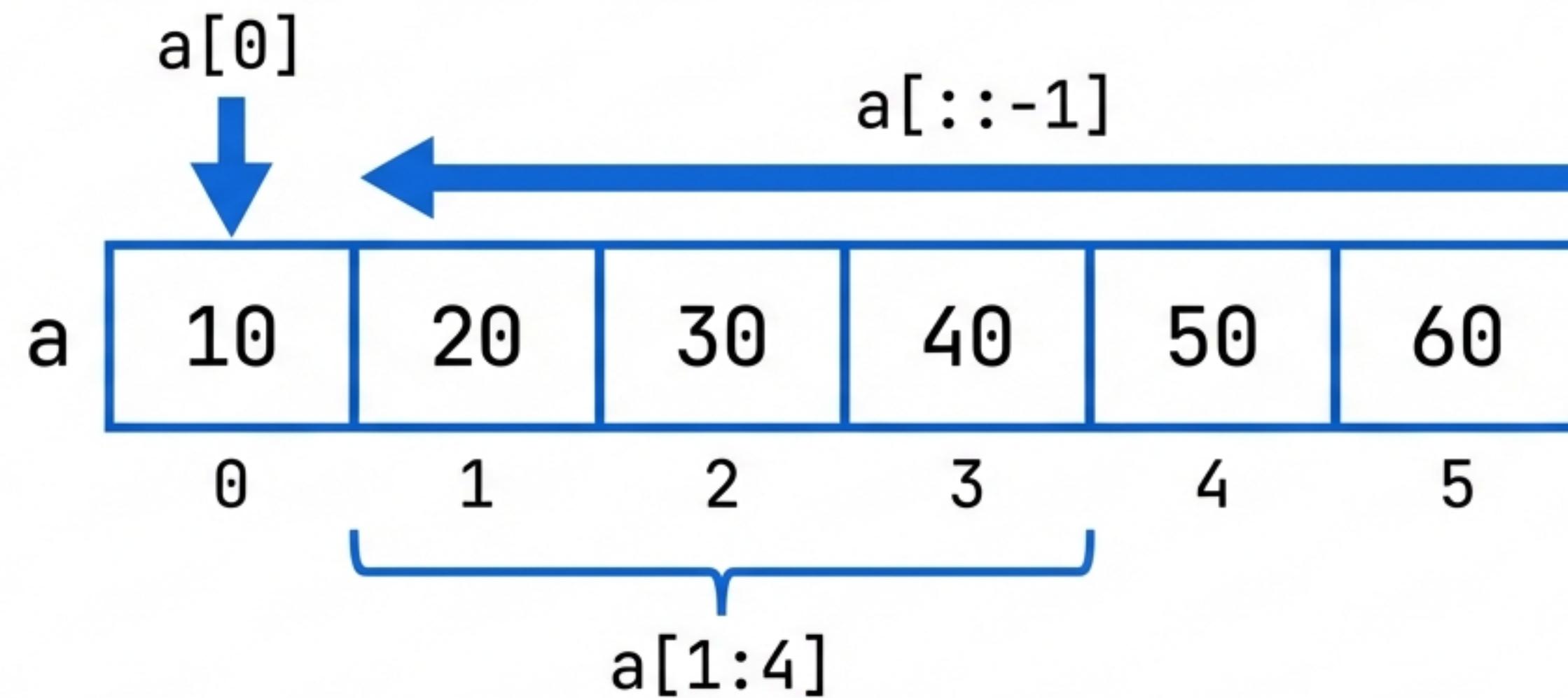
Niemals `math.sin()` auf Arrays anwenden!

Nutzen Sie die NumPy-Pendants für maximale Geschwindigkeit.

Kategorie	NumPy Funktion
Trigonometrie	<code>np.sin()</code> , <code>np.cos()</code> , <code>np.tan()</code>
Exponential/Log	<code>np.exp()</code> , <code>np.log()</code> , <code>np.log10()</code>
Sonstige	<code>np.abs()</code> , <code>np.sqrt()</code> , <code>np.sign()</code>



Indizierung und Slicing

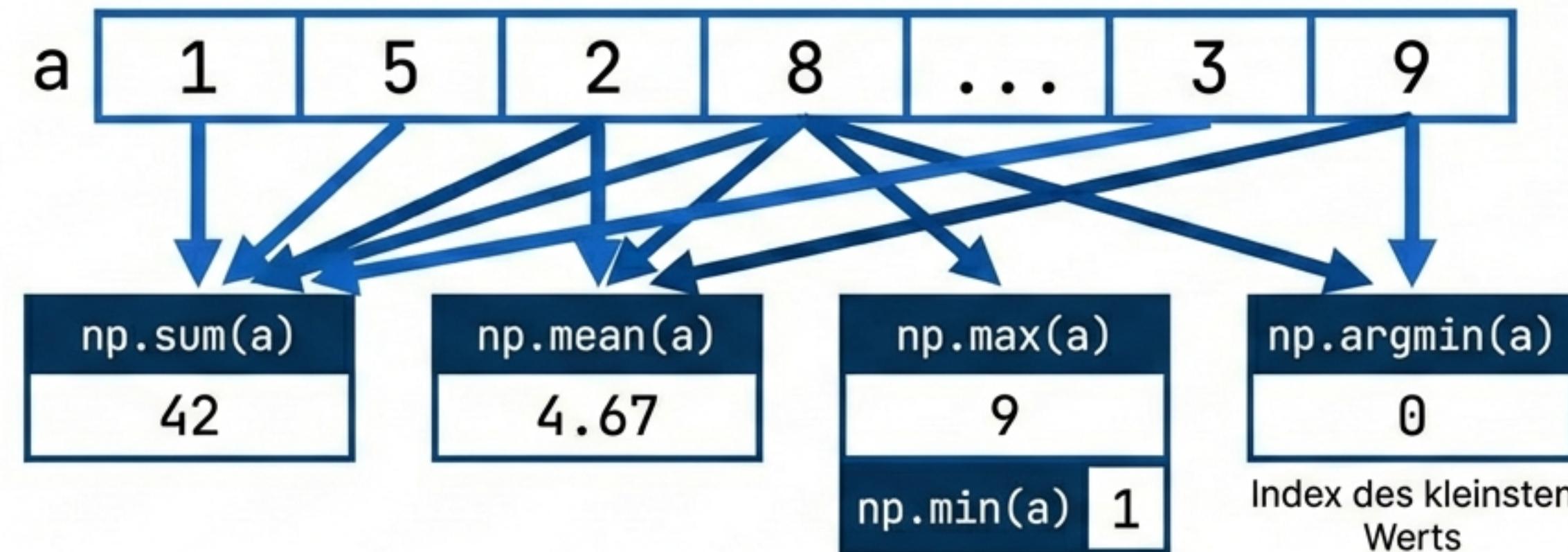


Achtung! Views vs. Copies

Slices sind in NumPy Views, keine Kopien. Eine Änderung im Slice ändert das Original-Array. Nutzen Sie `.copy()` um Daten wirklich zu duplizieren.

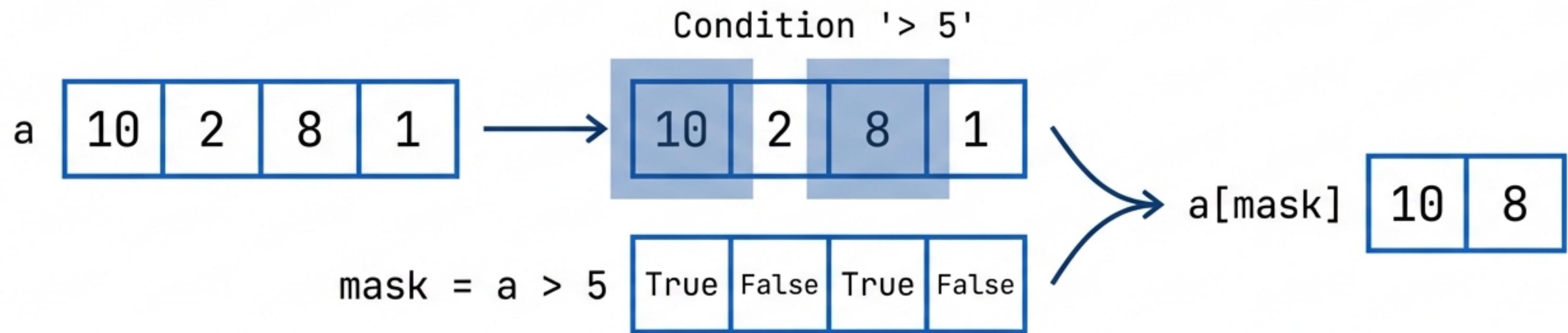
Aggregationen: Daten zusammenfassen

Extrem schnelle Berechnung statistischer Kennzahlen über das gesamte Array.



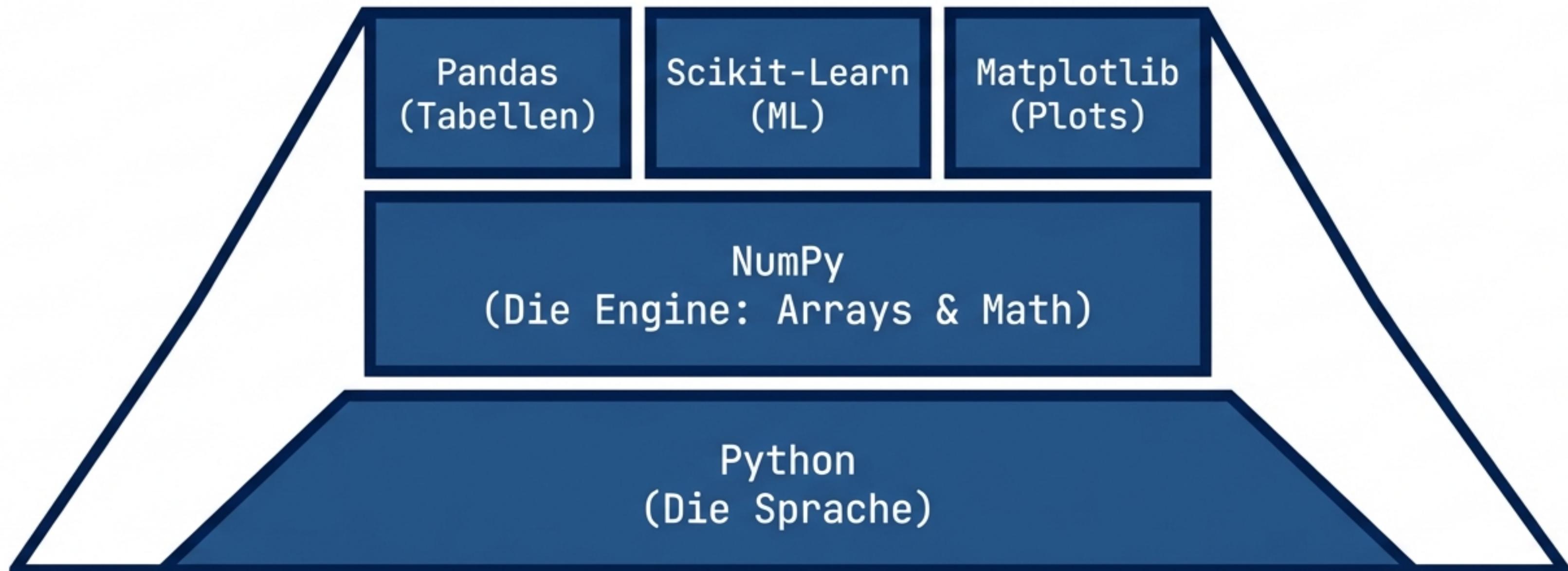
<code>np.sum(a)</code>	Summe aller Werte
<code>np.mean(a)</code>	Arithmetischer Mittelwert
<code>np.max(a) / np.min(a)</code>	Maximum / Minimum
<code>np.argmin(a)</code>	Index des kleinsten Werts

Filtern mit Boolescher Maskierung



Keine 'if'-Schleifen nötig, um Daten zu filtern.

Das NumPy-Ökosystem



Pandas DataFrames und Scikit-Learn Modelle basieren intern auf NumPy Arrays.
Wer NumPy versteht, versteht den Kern von Python Data Science.

Zusammenfassung

- ✓ **Performance:** Speicherlayout (Contiguous) und Vektorisierung machen NumPy massiv schneller.
- ✓ **Syntax:** Mathematische Operationen sind kompakt und lesbar ($a + b$).
- ✓ **1D-Arrays:** Die Basisstruktur für Vektoren und Zeitreihen.
- ✓ **Typisierung:** Homogene Datentypen (dtype) garantieren Effizienz.

NumPy ist der Standard für numerisches Rechnen in Python.

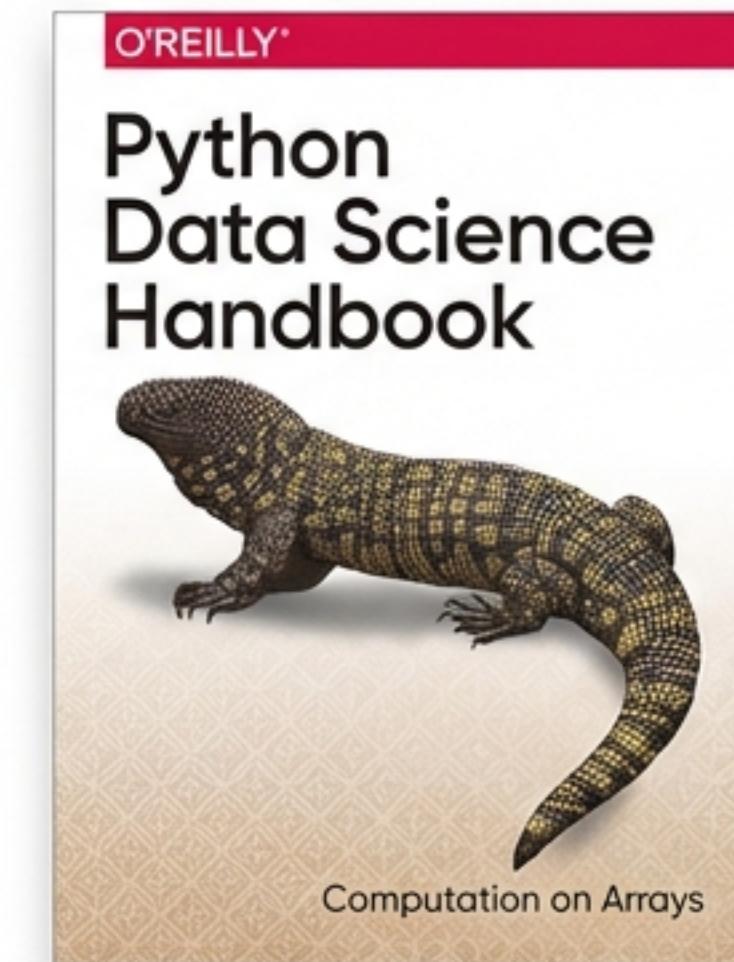
Weiterführende Ressourcen

Offizielle
Dokumentation

 numpy.org

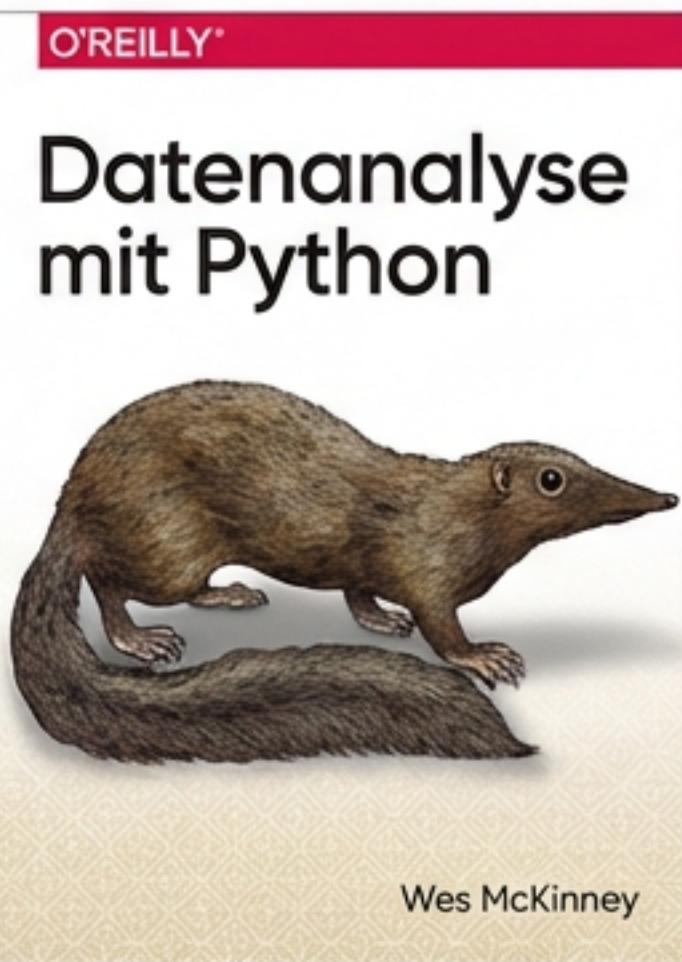
Python Data Science
Handbook

Kapitel: Computation on Arrays



Datenanalyse mit
Python

Wes McKinney



Viel Erfolg auf Ihrer Data Science Reise!